sourced.ml.core Documentation

Release master

Vadim Markovtsev

Oct 17, 2019

Contents

CHAPTER 1

sourced.ml.core

MLonCode research playground.

1.1 Subpackages

1.1.1 sourced.ml.core.algorithms

Subpackages

sourced.ml.core.algorithms.id_splitter

Submodules

sourced.ml.core.algorithms.id_splitter.features

Module Contents

sourced.ml.core.algorithms.id_splitter.features.**read_identifiers**(*csv_path: str*, *use_header: bool*, *max_identifier_len: int*, *identifier_col: int*, *split_identifier_col: int*, *shuffle: bool = True*)

Reads and filters too long identifiers in the CSV file.

Parameters

- **csv_path** – path to the CSV file.

- **use_header** – uses header as normal line (True) or treat as header line with column names.
- **max_identifier_len** – maximum length of raw identifiers. Skip identifiers that are longer.
- **identifier_col** – column name in the CSV file for the raw identifier.
- **split_identifier_col** – column name in the CSV file for the split identifier lowercase.
- **shuffle** – indicates whether to reorder the list of identifiers at random after reading it.

Returns list of split identifiers.

sourced.ml.core.algorithms.id_splitter.features.**prepare_features**(*csv_path: str*, *use_header: bool*, *max_identifier_len: int*, *identifier_col: int*, *split_identifier_col: int*, *test_ratio: float*, *padding: str*, *shuffle: bool = True*)

Prepare the features to train the identifier splitting task.

Parameters

- **csv_path** – path to the CSV file.
- **use_header** – uses header as normal line (True) or treat as header line with column names.
- **max_identifier_len** – maximum length of raw identifiers. Skip identifiers that are longer.
- **identifier_col** – column in the CSV file for the raw identifier.
- **split_identifier_col** – column in the CSV file for the split identifier.
- **shuffle** – indicates whether to reorder the list of identifiers at random after reading it.
- **test_ratio** – Proportion of test samples used for evaluation.
- **padding** – position where to add padding values: after the intput sequence if "post", before if "pre".

Returns training and testing features to train the neural net for the splitting task.

sourced.ml.core.algorithms.id_splitter.nn_model

Module Contents

sourced.ml.core.algorithms.id_splitter.nn_model.**LOSS = binary_crossentropy**

sourced.ml.core.algorithms.id_splitter.nn_model.**METRICS = ['accuracy']**

sourced.ml.core.algorithms.id_splitter.nn_model.**NUM_CHARS**

sourced.ml.core.algorithms.id_splitter.nn_model.**register_metric**(*metric: Union[str, Callable]*)

Decorator function to register the metrics in the METRICS constant.

Parameters **metric** – name of the tensorflow metric or custom function metric.

Returns the metric.

sourced.ml.core.algorithms.id_splitter.nn_model.**prepare_devices**(*devices: str*)

Extract devices from arguments.

Parameters **devices** – devices to use passed as one string argument.

Returns split devices.

sourced.ml.core.algorithms.id_splitter.nn_model.**prepare_input_emb**(*maxlen: int*)

Builds character embeddings, a dense representation of characters to feed the RNN with.

Parameters **maxlen** – maximum length of the input sequence.

Returns input and one-hot character embedding layer.

sourced.ml.core.algorithms.id_splitter.nn_model.**add_output_layer**(*hidden_layer: tf.Tensor*)

Applies a Dense layer to each of the timestamps of a hidden layer, independently. The output layer has 1 sigmoid per character which predicts if there is a space or not before the character.

Parameters **hidden_layer** – hidden layer before the output layer.

Returns output layer.

sourced.ml.core.algorithms.id_splitter.nn_model.**add_rnn**(*X: tf.Tensor*, *units: int*, *rnn_layer: str*, *dev0: str = '/gpu:0'*, *dev1: str = '/gpu:1'*)

Adds a bidirectional RNN layer with the specified parameters.

Parameters

- **X** – input layer.
- **units** – number of neurons in the output layer.
- **rnn_layer** – type of cell in the RNN.
- **dev0** – device that will be used as forward pass of RNN and concatenation.
- **dev1** – device that will be used as backward pass.

Returns output bidirectional RNN layer.

sourced.ml.core.algorithms.id_splitter.nn_model.**build_rnn**(*maxlen: int*, *units: int*, *stack: int*, *optimizer: str*, *dev0: str*, *dev1: str*, *rnn_layer: str*)

Builds a RNN model with the parameters specified as arguments.

Parameters

- **maxlen** – maximum length of the input sequence.
- **units** – number of neurons or dimensionality of the output RNN.
- **stack** – number of RNN layers to stack.

- **optimizer** – algorithm to use as an optimizer for the RNN.
- **rnn_layer** – recurrent layer type to use.
- **dev0** – first device to use when running specific operations.
- **dev1** – second device to use when running specific operations.

Returns compiled RNN model.

sourced.ml.core.algorithms.id_splitter.nn_model.**add_conv**(*X: tf.Tensor, filters: List[int], kernel_sizes: List[int], output_n_filters: int*)

Builds a single convolutional layer.

Parameters

- **X** – input layer.
- **filters** – number of output filters in the convolution.
- **kernel_sizes** – list of lengths of the 1D convolution window.
- **output_n_filters** – number of 1D output filters.

Returns output layer.

sourced.ml.core.algorithms.id_splitter.nn_model.**build_cnn**(*maxlen: int, filters: List[int], output_n_filters: int, stack: int, kernel_sizes: List[int], optimizer: str, device: str*)

Builds a CNN model with the parameters specified as arguments.

Parameters

- **maxlen** – maximum length of the input sequence.
- **filters** – number of output filters in the convolution.
- **output_n_filters** – number of 1d output filters.
- **stack** – number of CNN layers to stack.
- **kernel_sizes** – list of lengths of the 1D convolution window.
- **optimizer** – algorithm to use as an optimizer for the CNN.
- **device** – device to use when running specific operations.

Returns compiled CNN model.

sourced.ml.core.algorithms.id_splitter.nn_model.**precision**(*y_true: tf.Tensor, y_pred: tf.Tensor*)

Computes the precision, a metric for multi-label classification of how many selected items are relevant.

Parameters

- **y_true** – tensor of true labels.
- **y_pred** – tensor of predicted labels.

Returns a tensor batch-wise average of precision.

sourced.ml.core.algorithms.id_splitter.nn_model.**recall**(*y_true: tf.Tensor*, *y_pred: tf.Tensor*)

Computes the recall, a metric for multi-label classification of how many relevant items are selected.

Parameters

- **y_true** – tensor of true labels.
- **y_pred** – tensor of predicted labels.

Returns a tensor batch-wise average of recall.

sourced.ml.core.algorithms.id_splitter.nn_model.**f1score**(*y_true: tf.Tensor*, *y_pred: tf.Tensor*)

Computes the F1 score, the harmonic average of precision and recall.

Parameters

- **y_true** – tensor of true labels.
- **y_pred** – tensor of predicted labels.

Returns a tensor batch-wise average of F1 score.

sourced.ml.core.algorithms.id_splitter.pipeline

Module Contents

sourced.ml.core.algorithms.id_splitter.pipeline.**EPSILON**

sourced.ml.core.algorithms.id_splitter.pipeline.**DEFAULT_THRESHOLD = 0.5**

sourced.ml.core.algorithms.id_splitter.pipeline.**set_random_seed**(*seed: int*)

Fixes a random seed for reproducibility.

Parameters **seed** – seed value.

sourced.ml.core.algorithms.id_splitter.pipeline.**binarize**(*matrix: numpy.array*, *threshold: float*, *inplace: bool = True*)

Helper function to binarize a matrix.

Parameters

- **matrix** – matrix as a numpy.array.
- **threshold** – if value >= threshold then the value will be 1, else 0.
- **inplace** – whether to modify the matrix inplace or not.

Returns the binarized matrix.

sourced.ml.core.algorithms.id_splitter.pipeline.**str2ints**(*params: str*)

Convert a string with integer parameters to a list of integers.

Parameters **params** – string that contains integer parameters separated by commas.

Returns list of integers.

sourced.ml.core.algorithms.id_splitter.pipeline.**precision_np**(*y_true: numpy.array*, *y_pred: numpy.array*, *epsilon: float = EPSILON*)

Computes the precision metric, a metric for multi-label classification of how many selected items are relevant.

Parameters

- **y_true** – ground truth labels - expect binary values.
- **y_pred** – predicted labels - expect binary values.
- **epsilon** – added to the denominator to avoid any division by zero.

Returns precision metric.

sourced.ml.core.algorithms.id_splitter.pipeline.**recall_np**(*y_true: numpy.array*, *y_pred: numpy.array*, *epsilon: float = EPSILON*)

Computes the recall metric, a metric for multi-label classification of how many relevant items are selected.

Parameters

- **y_true** – matrix with ground truth labels - expect binary values.
- **y_pred** – matrix with predicted labels - expect binary values.
- **epsilon** – added to the denominator to avoid any division by zero.

Returns recall metric.

sourced.ml.core.algorithms.id_splitter.pipeline.**report**(*model: keras.engine.training.Model*, *X: numpy.array*, *y: numpy.array*, *batch_size: int*, *threshold: float = DEFAULT_THRESHOLD*, *epsilon: float = EPSILON*)

Prints a metric report of the *model* on the data *X* & *y*. The metrics printed are precision, recall, F1 score.

Parameters

- **model** – model considered.
- **X** – features.
- **y** – labels (expected binary labels).
- **batch_size** – batch size that will be used for prediction.
- **threshold** – threshold to binarize the predictions.
- **epsilon** – added to the denominator to avoid any division by zero.

sourced.ml.core.algorithms.id_splitter.pipeline.**config_keras**()

Initializes keras backend session.

sourced.ml.core.algorithms.id_splitter.pipeline.**build_train_generator**(*X: numpy.array*, *y: numpy.array*, *batch_size: int = 500*)

Builds the generator that yields features and their labels.

Parameters

- **X** – features.
- **y** – binary labels.
- **batch_size** – higher values better utilize GPUs.

Returns generator of features and their labels.

sourced.ml.core.algorithms.id_splitter.pipeline.**build_schedule**(*lr: float*, *final_lr: float*, *n_epochs: int*)

Builds the schedule of which the learning rate decreases. The schedule makes the learning rate decrease linearly.

Parameters

- **lr** – initial learning rate.
- **final_lr** – final learning rate.
- **n_epochs** – number of training epochs.

Returns the schedule of the learning rate.

sourced.ml.core.algorithms.id_splitter.pipeline.**make_lr_scheduler**(*lr: float*, *final_lr: float*, *n_epochs: int*, *verbose: int = 1*)

Prepares the scheduler to decrease the learning rate while training.

Parameters

- **lr** – initial learning rate.
- **final_lr** – final learning rate.
- **n_epochs** – number of training epochs.
- **verbose** – level of verbosity.

Returns LearningRateScheduler with linear schedule of the learning rate.

sourced.ml.core.algorithms.id_splitter.pipeline.**prepare_callbacks**(*output_dir: str*)

Prepares logging, tensorboard, model checkpoint callbacks and stores the outputs in output_dir.

Parameters **output_dir** – path to the results.

Returns list of callbacks.

sourced.ml.core.algorithms.id_splitter.pipeline.**create_generator_params**(*batch_size: int*, *samples_per_epoch: int*, *n_samples: int*, *epochs: int*)

Helper function to split a huge dataset into smaller ones to enable more frequent reports.

Parameters

- **batch_size** – batch size.
- **samples_per_epoch** – number of samples per mini-epoch or before each report.
- **n_samples** – total number of samples.
- **epochs** – number of epochs over the full dataset.

Returns number of steps per epoch (should be used with the generator) and number of sub-epochs where during sub-epoch only samples_per_epoch will be generated.

Submodules

sourced.ml.core.algorithms.id_embedding

Module Contents

sourced.ml.core.algorithms.id_embedding.**extract_coocc_matrix**(*global_shape*, *word_indices*, *model*)

sourced.ml.core.algorithms.swivel

Submatrix-wise Vector Embedding Learner.

Implementation of SwiVel algorithm described at: http://arxiv.org/abs/1602.02215

This program expects an input directory that contains the following files.

row_vocab.txt, col_vocab.txt

The row an column vocabulary files. Each file should contain one token per line; these will be used to generate a tab-separate file containing the trained embeddings.

row_sums.txt, col_sum.txt

The matrix row and column marginal sums. Each file should contain one decimal floating point number per line which corresponds to the marginal count of the matrix for that row or column.

shards.recs

A file containing the sub-matrix shards, stored as TFRecords. Each shard is expected to be a serialzed tf.Example protocol buffer with the following properties:

global_row: the global row indices contained in the shard global_col: the global column indices contained in the shard sparse_local_row, sparse_local_col, sparse_value: three parallel arrays that are a sparse representation of the submatrix counts.

It will generate embeddings, training from the input directory for the specified number of epochs. When complete, it will output the trained vectors to a tab-separated file that contains one line per embedding. Row and column embeddings are stored in separate files.

Module Contents

sourced.ml.core.algorithms.swivel.**flags**

sourced.ml.core.algorithms.swivel.**FLAGS**

sourced.ml.core.algorithms.swivel.**log**(*message*, **args*, ***kwargs*)

sourced.ml.core.algorithms.swivel.**get_available_gpus**()

sourced.ml.core.algorithms.swivel.**embeddings_with_init**(*vocab_size*, *embedding_dim*, *name*)

Creates and initializes the embedding tensors.

sourced.ml.core.algorithms.swivel.**count_matrix_input**(*filenames*, *submatrix_rows*, *submatrix_cols*)

Reads submatrix shards from disk.

sourced.ml.core.algorithms.swivel.**read_marginals_file**(*filename*)

Reads text file with one number per line to an array.

sourced.ml.core.algorithms.swivel.**write_embedding_tensor_to_disk**(*vocab_path*, *output_path*, *sess*, *embedding*)

Writes tensor to output_path as tsv

sourced.ml.core.algorithms.swivel.**write_embeddings_to_disk**(*config*, *model*, *sess*)

Writes row and column embeddings disk

class sourced.ml.core.algorithms.swivel.**SwivelModel**(*config*)

Small class to gather needed pieces from a Graph being built.

initialize_summary(*self*, *sess*)

write_summary(*self*, *sess*)

sourced.ml.core.algorithms.swivel.**main**(*_*)

sourced.ml.core.algorithms.tf_idf

Module Contents

sourced.ml.core.algorithms.tf_idf.**log_tf_log_idf**(*tf*, *df*, *ndocs*)

sourced.ml.core.algorithms.token_parser

Module Contents

class sourced.ml.core.algorithms.token_parser.**TokenStyle**

Bases:enum.Enum

Metadata that should allow to reconstruct initial identifier from a list of tokens.

DELIMITER = 1

TOKEN_UPPER = 2

TOKEN_LOWER = 3

TOKEN_CAPITALIZED = 4

class sourced.ml.core.algorithms.token_parser.**TokenParser**(*stem_threshold=STEM_THRESHOLD*, *max_token_length=MAX_TOKEN_LENGTH*, *min_split_length=MIN_SPLIT_LENGTH*, *single_shot=False*, *save_token_style=False*, *attach_upper=True*, *use_nn=False*, *nn_model=None*)

Common utilities for splitting and stemming tokens.

NAME_BREAKUP_RE

NAME_BREAKUP_KEEP_DELIMITERS_RE

STEM_THRESHOLD = 6

MAX_TOKEN_LENGTH = 256

MIN_SPLIT_LENGTH = 3

use_nn

stem_threshold

max_token_length

min_split_length

process_token(*self*, *token*)

stem(*self*, *word*)

split(*self*, *token: str*)
Splits a single identifier.

split_batch(*self, tokens: [str]*)
Splits a batch of identifiers.

static reconstruct(*tokens*)

class sourced.ml.core.algorithms.token_parser.**NoopTokenParser**
One can use this class one does not want to do any parsing.

process_token(*self*, *token*)

sourced.ml.core.algorithms.uast_id_distance

Module Contents

class sourced.ml.core.algorithms.uast_id_distance.**Uast2IdDistance**(*token2index=None*, *token_parser=None*, *max_distance=DEFAULT_MAX_DI*

Bases: *sourced.ml.core.algorithms.uast_ids_to_bag.UastIds2Bag*

Converts a UAST to a list of identifiers pair and UAST distance between. Distance metric must be defined in the inheritors.

__call__ is overridden here and return list instead of bag-of-words (dist).

DEFAULT_MAX_DISTANCE = 10

distance(*self*, *point1*, *point2*)

Calculate distance between two points. A point can be anything. self._process_uast returns list of points in the specific class.

Returns Distance between two points.

class sourced.ml.core.algorithms.uast_id_distance.**Uast2IdTreeDistance**

Bases: *sourced.ml.core.algorithms.uast_id_distance.Uast2IdDistance*

Converts a UAST to a list of identifiers pair and UAST tree distance between.

__call__ is overridden here and return list instead of bag-of-words (dist).

distance(*self*, *point1*, *point2*)

static calc_tree_distance(*last_common_level*, *level1*, *level2*)

class sourced.ml.core.algorithms.uast_id_distance.**Uast2IdLineDistance**

Bases: *sourced.ml.core.algorithms.uast_id_distance.Uast2IdDistance*

Converts a UAST to a list of identifiers pair and code line distance between where applicable.

__call__ is overridden here and return list instead of bag-of-words (dist).

distance(*self*, *point1*, *point2*)

sourced.ml.core.algorithms.uast_ids_to_bag

Module Contents

sourced.ml.core.algorithms.uast_ids_to_bag.**uast2sequence**(*root*)

class sourced.ml.core.algorithms.uast_ids_to_bag.**FakeVocabulary**

class sourced.ml.core.algorithms.uast_ids_to_bag.**UastTokens2Bag**(*token2index=None*, *token_parser=None*)

Bases: *sourced.ml.core.algorithms.uast_to_bag.Uast2BagBase*

Converts a UAST to a weighed bag of tokens via xpath.

XPATH

token_parser

token2index

class sourced.ml.core.algorithms.uast_ids_to_bag.**UastIds2Bag**(*token2index=None*, *token_parser=None*)

Bases: *sourced.ml.core.algorithms.uast_ids_to_bag.UastTokens2Bag*

Converts a UAST to a bag-of-identifiers.

XPATH = //*[@roleIdentifier]

sourced.ml.core.algorithms.uast_inttypes_to_graphlets

Module Contents

class sourced.ml.core.algorithms.uast_inttypes_to_graphlets.**Uast2GraphletBag**
Bases:sourced.ml.core.algorithms.uast_ids_to_bag.Uast2BagBase

Converts a UAST to a bag of graphlets. The graphlet of a UAST node is composed from the node itself, its parent and its children. Each node is represented by the internal role string.

uast2graphlets(*self*, *uast*)

Parameters **uast** – The UAST root node.

Generate The nodes which compose the UAST. :class: 'Node' is used to access the nodes of the graphlets.

node2key(*self*, *node*)
Builds the string joining internal types of all the nodes in the node's graphlet in the following order: parent_node_child1_child2_child3. The children are sorted by alphabetic order. str format is required for BagsExtractor.

Parameters **node** – a node of UAST

Returns The string key of node

sourced.ml.core.algorithms.uast_inttypes_to_nodes

Module Contents

class sourced.ml.core.algorithms.uast_inttypes_to_nodes.**Uast2QuantizedChildren**(*npartitions: int = 20*)
Bases:*sourced.ml.core.algorithms.uast_to_bag.Uast2BagThroughSingleScan*

Converts a UAST to a bag of children counts.

node2key(*self*, *node: bblfsh.Node*)
Return the key for a given Node.

Parameters **node** – a node in UAST.

Returns The string which consists of the internal type of the node and its number of children.

quantize(*self, frequencies: Iterable[Tuple[str, Iterable[Tuple[int, int]]]]*)

quantize_unwrapped(*self, children_freq: Iterable[Tuple[int, int]]*)
Builds the quantization partition P that is a vector of length nb_partitions whose entries are in strictly ascending order. Quantization of x is defined as:

0 if x <= P[0] m if P[m-1] < x <= P[m] n if P[n] <= x

Parameters **children_freq** – distribution of the number of children.

Returns The array with quantization levels.

sourced.ml.core.algorithms.uast_struct_to_bag

Module Contents

class sourced.ml.core.algorithms.uast_struct_to_bag.**Uast2StructBagBase**(*stride*, *seq_len*, *node2index=None*)

Bases: sourced.ml.core.algorithms.uast_ids_to_bag.Uast2BagBase

SEP = >

node2index

class sourced.ml.core.algorithms.uast_struct_to_bag.**Node2InternalType**

class sourced.ml.core.algorithms.uast_struct_to_bag.**UastSeq2Bag**(*stride=1*, *seq_len=(3*, *4)*, *node2index=None*)

Bases: *sourced.ml.core.algorithms.uast_struct_to_bag.Uast2StructBagBase*

DFS traversal + preserves the order of node children.

class sourced.ml.core.algorithms.uast_struct_to_bag.**Node**(*parent=None*, *internal_type=None*)

neighbours

class sourced.ml.core.algorithms.uast_struct_to_bag.**Uast2RandomWalks**(*p_explore_neighborhood*, *q_leave_neighborhood*, *n_walks*, *n_steps*, *node2index=None*, *seed=None*)

Generation of random walks for UAST.

prepare_starting_nodes(*self*, *uast*)

random_walk(*self*, *node*)

alias_sample(*self*, *walk*)

Compare to node2vec this sampling is a bit simpler because there is no loop in tree -> so there are only 2 options with unnormalized probabilities 1/p & 1/q Related article: https://arxiv.org/abs/1607.00653

Parameters **walk** – list of visited nodes

Returns next node to visit

class sourced.ml.core.algorithms.uast_struct_to_bag.**UastRandomWalk2Bag**(*p_explore_neighborhood=0.*
q_leave_neighborhood=0.82
n_walks=2, *n_steps=10*, *stride=1*, *seq_len=(2*, *3)*, *seed=42*)

Bases: *sourced.ml.core.algorithms.uast_struct_to_bag.Uast2StructBagBase*

sourced.ml.core.algorithms.uast_to_bag

Module Contents

class sourced.ml.core.algorithms.uast_to_bag.**Uast2BagBase**
Base class to convert UAST to a bag of anything.

class sourced.ml.core.algorithms.uast_to_bag.**Uast2BagThroughSingleScan**
Bases: *sourced.ml.core.algorithms.uast_to_bag.Uast2BagBase*

Constructs the bag by doing a single tree traversal and turning every node into a string.

node2key(*self*, *node*)

sourced.ml.core.algorithms.uast_to_id_sequence

Module Contents

class sourced.ml.core.algorithms.uast_to_id_sequence.**Uast2IdSequence**
Bases: *sourced.ml.core.algorithms.uast_id_distance.Uast2IdLineDistance*

Converts a UAST to a sorted sequence of identifiers. Identifiers are sorted by position in code. We do not change the order if positions are not present.

__call__ is overridden here and return list instead of bag-of-words (dist).

static concat(*id_sequence: Iterable*)

sourced.ml.core.algorithms.uast_to_role_id_pairs

Module Contents

class sourced.ml.core.algorithms.uast_to_role_id_pairs.**Uast2RoleIdPairs**(*token2index=None*, *token_parser=None*)
Bases: *sourced.ml.core.algorithms.uast_ids_to_bag.UastIds2Bag*

Converts a UAST to a list of pairs. Pair is identifier and role, where role is Node role where identifier was found.

__call__ is overridden here and returns list instead of bag-of-words (dist).

static merge_roles(*roles: Iterable[int]*)

Package Contents

sourced.ml.core.algorithms.**log_tf_log_idf**(*tf*, *df*, *ndocs*)

class sourced.ml.core.algorithms.**UastIds2Bag**(*token2index=None*, *token_parser=None*)
Bases: *sourced.ml.core.algorithms.uast_ids_to_bag.UastTokens2Bag*

Converts a UAST to a bag-of-identifiers.

XPATH = //*[@roleIdentifier]

sourced.ml.core.algorithms.**uast2sequence**(*root*)

class sourced.ml.core.algorithms.**UastRandomWalk2Bag**(*p_explore_neighborhood=0.79*, *q_leave_neighborhood=0.82*, *n_walks=2*, *n_steps=10*, *stride=1*, *seq_len=(2, 3)*, *seed=42*)

Bases: *sourced.ml.core.algorithms.uast_struct_to_bag.Uast2StructBagBase*

class sourced.ml.core.algorithms.**UastSeq2Bag**(*stride=1*, *seq_len=(3, 4)*, *node2index=None*)

Bases: *sourced.ml.core.algorithms.uast_struct_to_bag.Uast2StructBagBase*

DFS traversal + preserves the order of node children.

class sourced.ml.core.algorithms.**Uast2QuantizedChildren**(*npartitions: int = 20*)

Bases: *sourced.ml.core.algorithms.uast_to_bag.Uast2BagThroughSingleScan*

Converts a UAST to a bag of children counts.

node2key(*self*, *node: bblfsh.Node*)

Return the key for a given Node.

Parameters **node** – a node in UAST.

Returns The string which consists of the internal type of the node and its number of children.

quantize(*self, frequencies: Iterable[Tuple[str, Iterable[Tuple[int, int]]]]*)

quantize_unwrapped(*self, children_freq: Iterable[Tuple[int, int]]*)

Builds the quantization partition P that is a vector of length nb_partitions whose entries are in strictly ascending order. Quantization of x is defined as:

0 if x <= P[0] m if P[m-1] < x <= P[m] n if P[n] <= x

Parameters **children_freq** – distribution of the number of children.

Returns The array with quantization levels.

class sourced.ml.core.algorithms.**Uast2GraphletBag**

Bases: sourced.ml.core.algorithms.uast_ids_to_bag.Uast2BagBase

Converts a UAST to a bag of graphlets. The graphlet of a UAST node is composed from the node itself, its parent and its children. Each node is represented by the internal role string.

uast2graphlets(*self*, *uast*)

Parameters **uast** – The UAST root node.

Generate The nodes which compose the UAST. :class: 'Node' is used to access the nodes of the graphlets.

node2key(*self*, *node*)

Builds the string joining internal types of all the nodes in the node's graphlet in the following order: parent_node_child1_child2_child3. The children are sorted by alphabetic order. str format is required for BagsExtractor.

Parameters **node** – a node of UAST

Returns The string key of node

class sourced.ml.core.algorithms.**Uast2RoleIdPairs**(*token2index=None*, *token_parser=None*)

Bases: *sourced.ml.core.algorithms.uast_ids_to_bag.UastIds2Bag*

Converts a UAST to a list of pairs. Pair is identifier and role, where role is Node role where identifier was found.

__call__ is overridden here and returns list instead of bag-of-words (dist).

static merge_roles(*roles: Iterable[int]*)

class sourced.ml.core.algorithms.**Uast2IdLineDistance**
Bases: *sourced.ml.core.algorithms.uast_id_distance.Uast2IdDistance*

Converts a UAST to a list of identifiers pair and code line distance between where applicable.

__call__ is overridden here and return list instead of bag-of-words (dist).

distance(*self*, *point1*, *point2*)

class sourced.ml.core.algorithms.**Uast2IdTreeDistance**
Bases: *sourced.ml.core.algorithms.uast_id_distance.Uast2IdDistance*

Converts a UAST to a list of identifiers pair and UAST tree distance between.

__call__ is overridden here and return list instead of bag-of-words (dist).

distance(*self*, *point1*, *point2*)

static calc_tree_distance(*last_common_level*, *level1*, *level2*)

class sourced.ml.core.algorithms.**Uast2IdSequence**
Bases: *sourced.ml.core.algorithms.uast_id_distance.Uast2IdLineDistance*

Converts a UAST to a sorted sequence of identifiers. Identifiers are sorted by position in code. We do not change the order if positions are not present.

__call__ is overridden here and return list instead of bag-of-words (dist).

static concat(*id_sequence: Iterable*)

1.1.2 sourced.ml.core.extractors

Submodules

sourced.ml.core.extractors.bags_extractor

Module Contents

class sourced.ml.core.extractors.bags_extractor.**Extractor**
Bases: *sourced.ml.core.utils.pickleable_logger.PickleableLogger*

Converts a single UAST via *algorithm* to anything you need. It is a wrapper to use in *Uast2Features* Transformer in a pipeline.

NAME

ALGORITHM

OPTS

classmethod get_kwargs_fromcmdline(*cls*, *args*)

extract(*self*, *uast: bblfsh.Node*)

class sourced.ml.core.extractors.bags_extractor.**BagsExtractor**(*docfreq_threshold=None*, *weight=None*, ***kwargs*)
Bases: *sourced.ml.core.extractors.bags_extractor.Extractor*

Converts a single UAST into the weighted set (dictionary), where elements are strings and the values are floats. The derived classes must implement uast_to_bag().

DEFAULT_DOCFREQ_THRESHOLD = 5

NAMESPACE

OPTS

docfreq_threhold

ndocs

extract(*self*, *uast*)

uast_to_bag(*self*, *uast*)

class sourced.ml.core.extractors.bags_extractor.**RoleIdsExtractor**

Bases: *sourced.ml.core.extractors.bags_extractor.Extractor*

NAME = roleids

ALGORITHM

sourced.ml.core.extractors.children

Module Contents

class sourced.ml.core.extractors.children.**ChildrenBagExtractor**(*docfreq_threshold=None*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

Converts a UAST to the bag of pairs (internal type, quantized number of children).

NAME = children

NAMESPACE = c.

OPTS

npartitions

levels

extract(*self*, *uast*)

quantize(*self*, *frequencies: Iterable[Tuple[str, Iterable[Tuple[int, int]]]]*)

sourced.ml.core.extractors.graphlets

Module Contents

class sourced.ml.core.extractors.graphlets.**GraphletBagExtractor**(*docfreq_threshold=None*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

NAME = graphlet

NAMESPACE = g.

OPTS

uast_to_bag(*self*, *uast*)

sourced.ml.core.extractors.helpers

Module Contents

sourced.ml.core.extractors.helpers.**register_extractor**(*cls*)

sourced.ml.core.extractors.helpers.**get_names_from_kwargs**(*f*)

sourced.ml.core.extractors.helpers.**filter_kwargs**(*kwargs*, *func*)

sourced.ml.core.extractors.helpers.**create_extractors_from_args**(*args: argparse.Namespace*)

sourced.ml.core.extractors.id_sequence

Module Contents

class sourced.ml.core.extractors.id_sequence.**IdSequenceExtractor**(*split_stem=False*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

Extractor wrapper for Uast2RoleIdPairs algorithm. Note that this is unusual BagsExtractor since it returns iterable instead of bag.

The class did not wrap with @register_extractor because it does not produce bags as others do. So nobody outside code will see it or use it directly. For the same reason we a free to override NAMESPACE, NAME, OPTS fields with any value we want.

TODO(zurk): Split BagsExtractor into two clases: Extractor and BagsExtractor(Extractor), re-inherit this class from Extractor, delete explanations from docstring.

NAMESPACE =

NAME = id sequence

OPTS

extract(*self*, *uast: bblfsh.Node*)

sourced.ml.core.extractors.identifier_distance

Module Contents

class sourced.ml.core.extractors.identifier_distance.**IdentifierDistance**(*split_stem=False*, *type='tree'*, *max_distance=DEFAULT_*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

Extractor wrapper for Uast2IdTreeDistance and Uast2IdLineDistance algorithm. Note that this is an unusual BagsExtractor since it returns iterable instead of bag.

The class did not wrap with @register_extractor because it does not produce bags as others do. So nobody outside code will see it or use it directly. For the same reason we a free to override NAMESPACE, NAME, OPTS fields with any value we want.

TODO(zurk): Split BagsExtractor into two clases: Extractor and BagsExtractor(Extractor), re-inherit this class from Extractor, delete explanations from docstring.

class DistanceType

Tree = tree

Line = line

All

static resolve(*type*)

NAMESPACE =

NAME = Identifier distance

OPTS

DEFAULT_MAX_DISTANCE

extract(*self*, *uast: bblfsh.Node*)

sourced.ml.core.extractors.identifiers

Module Contents

class sourced.ml.core.extractors.identifiers.**IdentifiersBagExtractor**(*docfreq_threshold=None*, *split_stem=True*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

NAME = id

NAMESPACE = i.

OPTS

uast_to_bag(*self*, *uast*)

sourced.ml.core.extractors.literals

Module Contents

class sourced.ml.core.extractors.literals.**HashedTokenParser**

process_token(*self*, *token*)

class sourced.ml.core.extractors.literals.**Literals2Bag**(*token2index=None*, *token_parser=None*)

Bases: *sourced.ml.core.algorithms.uast_ids_to_bag.UastIds2Bag*

Converts a UAST to a bag-of-literals.

XPATH = //*[@roleLiteral]

class sourced.ml.core.extractors.literals.**LiteralsBagExtractor**(*docfreq_threshold=None*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

NAME = lit

NAMESPACE = l.

OPTS

uast_to_bag(*self*, *uast*)

sourced.ml.core.extractors.uast_random_walk

Module Contents

class sourced.ml.core.extractors.uast_random_walk.**UastRandomWalkBagExtractor**(*docfreq_threshold=*
***kwargs*)

Bases:sourced.ml.core.extractors.helpers.BagsExtractor

NAME = node2vec

NAMESPACE = r.

OPTS

uast_to_bag(*self*, *uast*)

sourced.ml.core.extractors.uast_seq

Module Contents

class sourced.ml.core.extractors.uast_seq.**UastSeqBagExtractor**(*docfreq_threshold=None*,
***kwargs*)

Bases:sourced.ml.core.extractors.helpers.BagsExtractor

NAME = uast2seq

NAMESPACE = s.

OPTS

uast_to_bag(*self*, *uast*)

Package Contents

sourced.ml.core.extractors.**get_names_from_kwargs**(*f*)

sourced.ml.core.extractors.**register_extractor**(*cls*)

sourced.ml.core.extractors.**filter_kwargs**(*kwargs*, *func*)

sourced.ml.core.extractors.**create_extractors_from_args**(*args: argparse.Namespace*)

class sourced.ml.core.extractors.**Extractor**

Bases:*sourced.ml.core.utils.pickleable_logger.PickleableLogger*

Converts a single UAST via *algorithm* to anything you need. It is a wrapper to use in *Uast2Features* Transformer in a pipeline.

NAME

ALGORITHM

OPTS

classmethod get_kwargs_fromcmdline(*cls*, *args*)

extract(*self*, *uast: bblfsh.Node*)

class sourced.ml.core.extractors.**BagsExtractor**(*docfreq_threshold=None*, *weight=None*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.Extractor*

Converts a single UAST into the weighted set (dictionary), where elements are strings and the values are floats. The derived classes must implement uast_to_bag().

DEFAULT_DOCFREQ_THRESHOLD = 5

NAMESPACE

OPTS

docfreq_threhold

ndocs

extract(*self*, *uast*)

uast_to_bag(*self*, *uast*)

class sourced.ml.core.extractors.**RoleIdsExtractor**

Bases: *sourced.ml.core.extractors.bags_extractor.Extractor*

NAME = roleids

ALGORITHM

class sourced.ml.core.extractors.**IdentifiersBagExtractor**(*docfreq_threshold=None*, *split_stem=True*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

NAME = id

NAMESPACE = i.

OPTS

uast_to_bag(*self*, *uast*)

class sourced.ml.core.extractors.**LiteralsBagExtractor**(*docfreq_threshold=None*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

NAME = lit

NAMESPACE = l.

OPTS

uast_to_bag(*self*, *uast*)

class sourced.ml.core.extractors.**UastRandomWalkBagExtractor**(*docfreq_threshold=None*, ***kwargs*)

Bases: sourced.ml.core.extractors.helpers.BagsExtractor

NAME = node2vec

NAMESPACE = r.

OPTS

uast_to_bag(*self*, *uast*)

class sourced.ml.core.extractors.**UastSeqBagExtractor**(*docfreq_threshold=None*, ***kwargs*)

Bases:sourced.ml.core.extractors.helpers.BagsExtractor

NAME = uast2seq

NAMESPACE = s.

OPTS

uast_to_bag(*self*, *uast*)

class sourced.ml.core.extractors.**ChildrenBagExtractor**(*docfreq_threshold=None*, ***kwargs*)

Bases:*sourced.ml.core.extractors.bags_extractor.BagsExtractor*

Converts a UAST to the bag of pairs (internal type, quantized number of children).

NAME = children

NAMESPACE = c.

OPTS

npartitions

levels

extract(*self*, *uast*)

quantize(*self, frequencies: Iterable[Tuple[str, Iterable[Tuple[int, int]]]]*)

class sourced.ml.core.extractors.**GraphletBagExtractor**(*docfreq_threshold=None*, ***kwargs*)

Bases:*sourced.ml.core.extractors.bags_extractor.BagsExtractor*

NAME = graphlet

NAMESPACE = g.

OPTS

uast_to_bag(*self*, *uast*)

class sourced.ml.core.extractors.**IdentifierDistance**(*split_stem=False*, *type='tree'*, *max_distance=DEFAULT_MAX_DISTANCE*, ***kwargs*)

Bases:*sourced.ml.core.extractors.bags_extractor.BagsExtractor*

Extractor wrapper for Uast2IdTreeDistance and Uast2IdLineDistance algorithm. Note that this is an unusual BagsExtractor since it returns iterable instead of bag.

The class did not wrap with @register_extractor because it does not produce bags as others do. So nobody outside code will see it or use it directly. For the same reason we a free to override NAMESPACE, NAME, OPTS fields with any value we want.

TODO(zurk): Split BagsExtractor into two clases: Extractor and BagsExtractor(Extractor), re-inherit this class from Extractor, delete explanations from docstring.

class DistanceType

Tree = tree

Line = line

All

static resolve(*type*)

NAMESPACE =

NAME = Identifier distance

OPTS

DEFAULT_MAX_DISTANCE

extract(*self*, *uast: bblfsh.Node*)

class sourced.ml.core.extractors.**IdSequenceExtractor**(*split_stem=False*, ***kwargs*)

Bases: *sourced.ml.core.extractors.bags_extractor.BagsExtractor*

Extractor wrapper for Uast2RoleIdPairs algorithm. Note that this is unusual BagsExtractor since it returns iterable instead of bag.

The class did not wrap with @register_extractor because it does not produce bags as others do. So nobody outside code will see it or use it directly. For the same reason we a free to override NAMESPACE, NAME, OPTS fields with any value we want.

TODO(zurk): Split BagsExtractor into two clases: Extractor and BagsExtractor(Extractor), re-inherit this class from Extractor, delete explanations from docstring.

NAMESPACE =

NAME = id sequence

OPTS

extract(*self*, *uast: bblfsh.Node*)

1.1.3 sourced.ml.core.models

Subpackages

sourced.ml.core.models.model_converters

Submodules

sourced.ml.core.models.model_converters.base

Module Contents

class sourced.ml.core.models.model_converters.base.**Model2Base**(*num_processes: int = 0*, *log_level: int = logging.DEBUG*, *overwrite_existing: bool = True*)

Bases: *sourced.ml.core.utils.pickleable_logger.PickleableLogger*

Base class for model -> model conversions.

MODEL_FROM_CLASS

MODEL_TO_CLASS

convert (*self, models_path: List[str], destdir: str*)
Performs the model -> model conversion. Runs the conversions in a pool of processes.

Parameters

- **models_path** – List of Models path.
- **destdir** – The directory where to store the models. The directory structure is preserved.

Returns The number of converted files.

convert_model (*self*, *model: Model*)
This must be implemented in the child classes.

Parameters **model** – The model instance to convert.

Returns The converted model instance or None if it is not needed.

finalize (*self*, *index: int*, *destdir: str*)
Called for each worker in the end of the processing.

Parameters

- **index** – Worker's index.
- **destdir** – The directory where to store the models.

sourced.ml.core.models.model_converters.merge_bow

Module Contents

class sourced.ml.core.models.model_converters.merge_bow.**MergeBOW** (*features=None*, **args*, ***kwargs*)

Bases: *sourced.ml.core.models.model_converters.base.Model2Base*

Merges several BOW models together.

MODEL_FROM_CLASS

MODEL_TO_CLASS

convert_model (*self*, *model: BOW*)

finalize (*self*, *index: int*, *destdir: str*)

sourced.ml.core.models.model_converters.merge_df

Module Contents

class sourced.ml.core.models.model_converters.merge_df.**MergeDocFreq**(*min_docfreq: int, vocabulary_size: int, ordered: bool = False, *args, **kwargs*)

Bases: *sourced.ml.core.models.model_converters.base.Model2Base*

Merges several `DocumentFrequencies` models together.

MODEL_FROM_CLASS

MODEL_TO_CLASS

convert_model(*self*, *model: DocumentFrequencies*)

finalize(*self*, *index: int*, *destdir: str*)

Submodules

sourced.ml.core.models.bow

Module Contents

class sourced.ml.core.models.bow.**BOW**

Bases: `modelforge.Model`

Weighted bag of words model. Every word is correspond to an index and its matrix column. Bag is a word set from repository, file or anything else. Word is source code identifier or its part. This model depends on `sourced.ml.models.DocumentFrequencies`.

NAME = bow

VENDOR = source{d}

DESCRIPTION = Model that contains source code as weighted bag of words.

LICENSE

matrix
Returns the bags as a sparse matrix. Rows are documents and columns are tokens weight.

documents
The list of documents in the model.

tokens
The list of tokens in the model.

construct(*self, documents: List[str], tokens: List[str], matrix: sparse.spmatrix*)

dump(*self*)

save(*self*, *output: str*, *series: str*, *deps: Iterable = tuple()*, *create_missing_dirs: bool = True*)

convert_bow_to_vw(*self*, *output: str*)

documents_index(*self*)

sourced.ml.core.models.coocc

Module Contents

class sourced.ml.core.models.coocc.**Cooccurrences**
Bases:modelforge.model.Model

Co-occurrence matrix.

NAME = co-occurrences

VENDOR = source{d}

DESCRIPTION = Model that contains the sparse co-occurrence matrix of source code ident

LICENSE

tokens
Returns the tokens in the order which corresponds to the matrix's rows and cols.

matrix
Returns the sparse co-occurrence matrix.

construct(*self*, *tokens*, *matrix*)

dump(*self*)

matrix_to_rdd(*self*, *spark_context: 'pyspark.SparkContext'*)

sourced.ml.core.models.df

Module Contents

class sourced.ml.core.models.df.**DocumentFrequencies**
Bases:modelforge.Model

Document frequencies - number of times a source code identifier appeared in different repositories. Each repository counts only once.

NAME = docfreq

VENDOR = source{d}

DESCRIPTION = Model that contains document frequencies of features extracted from code

LICENSE

docs
Returns the number of documents.

construct(*self, docs: int, tokfreqs: Union[Iterable[Dict[str, int]], Dict[str, int]]*)
Initializes this model.

Parameters

- **docs** – The number of documents.
- **tokfreqs** – The dictionary of token -> frequency or the iterable collection of such dictionaries.

Returns self

dump (*self*)

prune (*self*, *threshold: int*)
Removes tokens which occur less than *threshold* times. The operation happens *not* in-place - a new model is returned. :param threshold: Minimum number of occurrences. :return: The new model if the current one had to be changed, otherwise self.

greatest (*self*, *max_size: int*)
Truncates the model to most frequent *max_size* tokens. The operation happens *not* in-place - a new model is returned. :param max_size: The maximum vocabulary size. :return: The new model if the current one had to be changed, otherwise self.

get (*self*, *item*, *default=None*)
Return the document frequency for a given token.

Parameters

- **item** – The token to query.
- **default** – Returned value in case the token is missing.

Returns int or *default*

tokens (*self*)
Returns the list of tokens.

sourced.ml.core.models.id2vec

Module Contents

class sourced.ml.core.models.id2vec.**Id2Vec**
Bases:modelforge.Model

id2vec model - source code identifier embeddings.

NAME = id2vec

VENDOR = source{d}

DESCRIPTION = Model that contains information on source code as identifier embeddings.

LICENSE

embeddings
numpy.ndarray with the embeddings of shape (N tokens x embedding dims).

tokens
List with the processed source code identifiers.

construct (*self*, *embeddings*, *tokens*)

dump (*self*)

items (*self*)
Returns the tuples belonging to token -> index mapping.

sourced.ml.core.models.id_splitter

Module Contents

class sourced.ml.core.models.id_splitter.**IdentifierSplitterBiLSTM**(***kwargs*)
Bases:modelforge.Model

Bidirectional LSTM Model. Splits identifiers without need for a conventional pattern. Reference: https://arxiv.org/abs/1805.11651

NAME = id_splitter_bilstm

VENDOR = source{d}

DESCRIPTION = Weights of the BiLSTM network to split source code identifiers.

LICENSE

DEFAULT_MAXLEN = 40

DEFAULT_PADDING = post

DEFAULT_MAPPING

DEFAULT_BATCH_SIZE = 4096

model
Return the wrapped keras model.

batch_size
Return the batch size used to run the model.

construct(*self*, *model: keras.models.Model*, *maxlen: int = DEFAULT_MAXLEN*, *padding: str = DEFAULT_PADDING*, *mapping: Dict[str, int] = DEFAULT_MAPPING*, *batch_size: int = DEFAULT_BATCH_SIZE*)
Construct IdentifierSplitterBiLSTM model.

Parameters

- **model** – keras model used for identifier splitting.
- **maxlen** – Maximum length of input identifers.
- **padding** – Where to pad the identifiers of length < maxlen. Can be "left" or "right".
- **mapping** – Mapping of characters to integers.
- **batch_size** – Batch size of input data fed to the model.

Returns BiLSTM based source code identifier splitter.

dump(*self*)

prepare_input(*self, identifiers: Sequence[str]*)
Prepare input by converting a sequence of identifiers to the corresponding ascii code 2D-array and the list of lowercase cleaned identifiers.

load_model_file(*self*, *path: str*)
Load a compatible Keras model file. Used for compatibility.

split(*self, identifiers: Sequence[str]*)
Split identifiers in a list, using the model.

sourced.ml.core.models.license

Default license used for the models.

Module Contents

sourced.ml.core.models.license.**DEFAULT_LICENSE = ODbL-1.0**

sourced.ml.core.models.ordered_df

Module Contents

class sourced.ml.core.models.ordered_df.**OrderedDocumentFrequencies**
Bases:*sourced.ml.core.models.DocumentFrequencies*

Compatible with the original DocumentFrequencies. This model maintains the determinitic sequence of the tokens.

order

construct (*self, docs: int, tokfreqs: Iterable[Dict[str, int]]*)

tokens (*self*)

prune (*self*, *threshold: int*)

greatest (*self*, *max_size: int*)

sourced.ml.core.models.quant

Module Contents

class sourced.ml.core.models.quant.**QuantizationLevels**
Bases:modelforge.Model

This model contains quantization levels for multiple schemes (feature types). Every feature "class" (type, possible distinct value) corresponds to the numpy array with integer level borders. The size of each numpy array is (the number of levels + 1).

NAME = quant

VENDOR = source{d}

DESCRIPTION = Model that contains quantization levels for multiple schemes (feature ty

LICENSE

levels

construct (*self, levels: Dict[str, Dict[str, numpy.ndarray]]*)

dump (*self*)

apply_quantization (*self*, *extractors*)

sourced.ml.core.models.tensorflow

Module Contents

class sourced.ml.core.models.tensorflow.**TensorFlowModel**
Bases:modelforge.Model

TensorFlow Protobuf model exported in the Modelforge format with GraphDef inside.

NAME = tensorflow-model

VENDOR = source{d}

DESCRIPTION = TensorFlow Protobuf model that contains a GraphDef instance.

LICENSE

graphdef
Returns the wrapped TensorFlow GraphDef.

construct(*self*, *graphdef: 'tensorflow.GraphDef' = None*, *session: 'tensorflow.Session' = None*, *outputs: List[str] = None*)

sourced.ml.core.models.topics

Module Contents

class sourced.ml.core.models.topics.**Topics**
Bases:modelforge.Model

NAME = topics

VENDOR = source{d}

DESCRIPTION = Model that is used to identify topics of source code repositories.

LICENSE

tokens

topics
May be None if no topics are labeled.

matrix
Rows: tokens Columns: topics

construct(*self, tokens: list, topics: Union[list, None], matrix*)

dump(*self*)

label_topics(*self*, *labels*)

Package Contents

class sourced.ml.core.models.**BOW**
Bases:modelforge.Model

Weighted bag of words model. Every word is correspond to an index and its matrix column. Bag is a word set from repository, file or anything else. Word is source code identifier or its part. This model depends on sourced.ml.models.DocumentFrequencies.

NAME = bow

VENDOR = source{d}

DESCRIPTION = Model that contains source code as weighted bag of words.

LICENSE

matrix
Returns the bags as a sparse matrix. Rows are documents and columns are tokens weight.

documents
The list of documents in the model.

tokens
The list of tokens in the model.

construct(*self, documents: List[str], tokens: List[str], matrix: sparse.spmatrix*)

dump(*self*)

save(*self*, *output: str*, *series: str*, *deps: Iterable = tuple()*, *create_missing_dirs: bool = True*)

convert_bow_to_vw(*self*, *output: str*)

documents_index(*self*)

class sourced.ml.core.models.**Cooccurrences**
Bases:modelforge.model.Model

Co-occurrence matrix.

NAME = co-occurrences

VENDOR = source{d}

DESCRIPTION = Model that contains the sparse co-occurrence matrix of source code ident

LICENSE

tokens
Returns the tokens in the order which corresponds to the matrix's rows and cols.

matrix
Returns the sparse co-occurrence matrix.

construct(*self*, *tokens*, *matrix*)

dump(*self*)

matrix_to_rdd(*self*, *spark_context: 'pyspark.SparkContext'*)

class sourced.ml.core.models.**DocumentFrequencies**
Bases:modelforge.Model

Document frequencies - number of times a source code identifier appeared in different repositories. Each repository counts only once.

NAME = docfreq

VENDOR = source{d}

DESCRIPTION = Model that contains document frequencies of features extracted from code

LICENSE

docs
Returns the number of documents.

construct (*self, docs: int, tokfreqs: Union[Iterable[Dict[str, int]], Dict[str, int]]*)
Initializes this model.

Parameters

- **docs** – The number of documents.
- **tokfreqs** – The dictionary of token -> frequency or the iterable collection of such dictionaries.

Returns self

dump (*self*)

prune (*self*, *threshold: int*)
Removes tokens which occur less than *threshold* times. The operation happens *not* in-place - a new model is returned. :param threshold: Minimum number of occurrences. :return: The new model if the current one had to be changed, otherwise self.

greatest (*self*, *max_size: int*)
Truncates the model to most frequent *max_size* tokens. The operation happens *not* in-place - a new model is returned. :param max_size: The maximum vocabulary size. :return: The new model if the current one had to be changed, otherwise self.

get (*self*, *item*, *default=None*)
Return the document frequency for a given token.

Parameters

- **item** – The token to query.
- **default** – Returned value in case the token is missing.

Returns int or *default*

tokens (*self*)
Returns the list of tokens.

class sourced.ml.core.models.**OrderedDocumentFrequencies**
Bases: *sourced.ml.core.models.DocumentFrequencies*

Compatible with the original DocumentFrequencies. This model maintains the determinitic sequence of the tokens.

order

construct (*self, docs: int, tokfreqs: Iterable[Dict[str, int]]*)

tokens (*self*)

prune (*self*, *threshold: int*)

greatest (*self*, *max_size: int*)

class sourced.ml.core.models.**Id2Vec**
Bases: modelforge.Model

id2vec model - source code identifier embeddings.

NAME = id2vec

VENDOR = source{d}

DESCRIPTION = Model that contains information on source code as identifier embeddings.

LICENSE

embeddings
numpy.ndarray with the embeddings of shape (N tokens x embedding dims).

tokens
List with the processed source code identifiers.

construct(*self*, *embeddings*, *tokens*)

dump(*self*)

items(*self*)
Returns the tuples belonging to token -> index mapping.

class sourced.ml.core.models.**TensorFlowModel**
Bases:modelforge.Model

TensorFlow Protobuf model exported in the Modelforge format with GraphDef inside.

NAME = tensorflow-model

VENDOR = source{d}

DESCRIPTION = TensorFlow Protobuf model that contains a GraphDef instance.

LICENSE

graphdef
Returns the wrapped TensorFlow GraphDef.

construct(*self*, *graphdef: 'tensorflow.GraphDef' = None*, *session: 'tensorflow.Session' = None*, *outputs: List[str] = None*)

class sourced.ml.core.models.**Topics**
Bases:modelforge.Model

NAME = topics

VENDOR = source{d}

DESCRIPTION = Model that is used to identify topics of source code repositories.

LICENSE

tokens

topics
May be None if no topics are labeled.

matrix
Rows: tokens Columns: topics

construct(*self, tokens: list, topics: Union[list, None], matrix*)

dump(*self*)

label_topics(*self*, *labels*)

class sourced.ml.core.models.**QuantizationLevels**
Bases:modelforge.Model

This model contains quantization levels for multiple schemes (feature types). Every feature "class" (type, possible distinct value) corresponds to the numpy array with integer level borders. The size of each numpy array is (the number of levels + 1).

NAME = quant

VENDOR = source{d}

DESCRIPTION = Model that contains quantization levels for multiple schemes (feature ty

LICENSE

levels

construct(*self*, *levels: Dict[str, Dict[str, numpy.ndarray]]*)

dump(*self*)

apply_quantization(*self*, *extractors*)

class sourced.ml.core.models.**MergeDocFreq**(*min_docfreq: int*, *vocabulary_size: int*, *ordered: bool = False*, **args*, ***kwargs*)

Bases: *sourced.ml.core.models.model_converters.base.Model2Base*

Merges several *DocumentFrequencies* models together.

MODEL_FROM_CLASS

MODEL_TO_CLASS

convert_model(*self*, *model: DocumentFrequencies*)

finalize(*self*, *index: int*, *destdir: str*)

class sourced.ml.core.models.**MergeBOW**(*features=None*, **args*, ***kwargs*)

Bases: *sourced.ml.core.models.model_converters.base.Model2Base*

Merges several *BOW* models together.

MODEL_FROM_CLASS

MODEL_TO_CLASS

convert_model(*self*, *model: BOW*)

finalize(*self*, *index: int*, *destdir: str*)

1.1.4 sourced.ml.core.utils

Submodules

sourced.ml.core.utils.bblfsh

Module Contents

sourced.ml.core.utils.bblfsh.**BBLFSH_VERSION_LOW = 2.2**

sourced.ml.core.utils.bblfsh.**BBLFSH_VERSION_HIGH = 3.0**

sourced.ml.core.utils.bblfsh.**check_version**(*host: str = '0.0.0.0'*, *port: str = '9432'*)

Check if the bblfsh server version matches module requirements.

Parameters

- **host** – bblfsh server host.
- **port** – bblfsh server port.

Returns True if bblfsh version specified matches requirements.

sourced.ml.core.utils.bblfsh_roles

Module Contents

sourced.ml.core.utils.bblfsh_roles.**IDENTIFIER**

sourced.ml.core.utils.bblfsh_roles.**QUALIFIED**

sourced.ml.core.utils.bblfsh_roles.**LITERAL**

sourced.ml.core.utils.bblfsh_roles.**OPERATOR**

sourced.ml.core.utils.bblfsh_roles.**EXPRESSION**

sourced.ml.core.utils.bblfsh_roles.**LEFT**

sourced.ml.core.utils.bblfsh_roles.**BINARY**

sourced.ml.core.utils.bblfsh_roles.**ASSIGNMENT**

sourced.ml.core.utils.bblfsh_roles.**FUNCTION**

sourced.ml.core.utils.bblfsh_roles.**DECLARATION**

sourced.ml.core.utils.bblfsh_roles.**NAME**

sourced.ml.core.utils.bigartm

Module Contents

sourced.ml.core.utils.bigartm.**execute**(*cmd*, *cwd*, *log*)

sourced.ml.core.utils.bigartm.**install_bigartm**(*args=None*, *target='./bigartm'*, *tempdir=None*)

Deploys bigartm/bigartm at the specified path.

Parameters

- **args** – argparse.Namespace with "output" and "tmpdir". "output" sets the target directory, "tmpdir" sets the temporary directory which is used to clone bigartm/bigartm and build it.
- **target** – The path to the built executable. If args is not None, it becomes overridden.
- **tempdir** – The temporary directory where to clone and build bigartm/bigartm. If args is not None, it becomes overridden.

Returns None if successful; otherwise, the error code (can be 0!).

sourced.ml.core.utils.pickleable_logger

Module Contents

class sourced.ml.core.utils.pickleable_logger.**PickleableLogger**(*log_level=logging.INFO*)

Base class which provides the logging features through `self._log`.

Can be safely pickled.

sourced.ml.core.utils.projector

Module Contents

class sourced.ml.core.utils.projector.**CORSWebServer**

running

serve(*self*)

start(*self*)

stop(*self*)

sourced.ml.core.utils.projector.**web_server**

sourced.ml.core.utils.projector.**present_embeddings**(*destdir*, *run_server*, *labels*, *index*, *embeddings*)

sourced.ml.core.utils.projector.**wait**()

Package Contents

sourced.ml.core.utils.**install_bigartm**(*args=None*, *target='./bigartm'*, *tempdir=None*)
Deploys bigartm/bigartm at the specified path.

Parameters

- **args** – argparse.Namespace with "output" and "tmpdir". "output" sets the target directory, "tmpdir" sets the temporary directory which is used to clone bigartm/bigartm and build it.
- **target** – The path to the built executable. If args is not None, it becomes overridden.
- **tempdir** – The temporary directory where to clone and build bigartm/bigartm. If args is not None, it becomes overridden.

Returns None if successful; otherwise, the error code (can be 0!).

class sourced.ml.core.utils.**PickleableLogger**(*log_level=logging.INFO*)
Base class which provides the logging features through self._log.

Can be safely pickled.

1.2 Submodules

1.2.1 sourced.ml.core.modelforgecfg

Module Contents

sourced.ml.core.modelforgecfg.**VENDOR = source{d}**

sourced.ml.core.modelforgecfg.**BACKEND = gcs**

sourced.ml.core.modelforgecfg.**BACKEND_ARGS = bucket=models.cdn.sourced.tech**

sourced.ml.core.modelforgecfg.**INDEX_REPO = https://github.com/src-d/models**

sourced.ml.core.modelforgecfg.**CACHE_DIR**

1.3 Package Contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X